ASSOCIATING IDENTIFIERS WITH VIRTUAL PROCESSES

INVENTORS

Pawan Goyal, Snorri Gylfason, Xun Wilson Huang, Srinivasan Keshav and Rosen Sharma

BACKGROUND

FIELD OF INVENTION

The present invention relates generally to virtual computer processes, and specifically to associating an identifier with each of a plurality of processes comprising a virtual process.

BACKGROUND OF INVENTION

With the popularity and success of the Internet, server technologies are of great commercial importance today. Typically, an individual server application executes on a single physical host computer, servicing client requests. However, providing a unique physical host for each server application is expensive and inefficient. Hosting services are often provided commercially by an Internet Service Provider (ISP). Typically, an ISP has to provide a separate physical host computer on which to execute a server application for every customer that purchases hosting services. Often, a customer purchasing hosting

services from an ISP will neither require nor be amenable to paying for use of an entire host computer. Generally, only a fraction of the processing power, storage, and other resources of a host computer will be required to meet the needs of an individual customer.

It is desirable for an ISP to be able to provide multiple, server applications on a single physical host computer. However, in order to be commercially viable, every server application would have to be isolated from every other server application running on the same physical host. Obviously, it would unacceptable to customers of an ISP to purchase hosting services, only to have another server application program (perhaps belonging to a competitor) be able to access the customer's data and client requests. Thus, each server application program would have to be isolated, receiving only requests from its own clients, transmitting data only to its own clients, and being prevented from accessing data associated with other server applications. Furthermore, it would be necessary to allocate varying specific levels of system resources to different server applications, depending upon the needs of and amounts paid by the various customers of the ISP. In effect, each server application would need to comprise a virtual private server, simulating a server application executing on a dedicated physical host computer.

Such functionality has been heretofore impossible because a virtual private server, rather than comprising a single, discrete process, must be made up of a plurality of seemingly unrelated processes, each performing various elements of the sum total of the functionality required by the customer. Because each virtual private server must comprise a plurality of processes, it has been impossible for an ISP to isolate the processes associated with one virtual private server from those processes associated with

other virtual private servers. What is needed is a method [illegible] pluralities of separate processes comprising multiple, virtual private servers, individual ones of associated with their respective virtual private server. [illegible] can each be

SUMMARY OF INVENTION

The present invention facilitates the association of multiple processes with their respective virtual private servers. In order to run multiple virtual processes on a single, physical computer system, each virtual private server is started by executing a separate, system initialization process. Each system initialization process is associated with a virtual private server identifier, for example by storing the identification number of the process and the virtual private server identifier in a data structure in computer memory. Each virtual private server identifier may be associated with a particular customer to identify the virtual private server belonging to the customer.

System calls that create processes are intercepted, and a system call wrapper associates created processes with the virtual private server identifier with which the process that made the system call is associated. Thus, all processes originating from each system initialization process will be associated with the virtual private server identifier associated with the corresponding system initialization process. Because all processes that are part of a virtual private server will be created by the associated system initialization process or by its progeny, all processes comprising a virtual private server will be associated with the corresponding virtual private server identifier. This allows all processes that are a part of each customer's virtual private server to be identified and

segregated from the processes of other customers, even though these processes are executing on the same physical server.

It will be readily apparent to one skilled in the art that the present invention can be utilized to associate a plurality of processes comprising any type of virtual process with a corresponding virtual process identifier. Of course, all such utilizations are within the scope of the present invention. Although in one embodiment, the virtual process is in the form of a virtual private server, the present invention is by no means limited to this embodiment.

The features and advantages described in this summary and the following detailed description are not all-inclusive, and particularly, many additional features and advantages will be apparent to one of ordinary skill in the art in view of the drawings, specification, and claims hereof. Moreover, it should be noted that the language used in the specification has been principally selected for readability and instructional purposes, and may not have been selected to delineate or circumscribe the inventive subject matter, resort to the claims being necessary to determine such inventive subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram providing a high level overview of a system for associating identifiers with virtual processes, according to one embodiment of the present invention.

FIG. 2 is a block diagram illustrating a system utilizing a virtual process manager program, according to one embodiment of the present invention.

FIG. 3 is a block diagram illustrating a system utilizing a modified loader program, according to another embodiment of the present invention.

The figures depict embodiments of the present invention for purposes of illustration only. One skilled in the art will readily recognize from the following discussion that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the invention described herein.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

I. SYSTEM OVERVIEW

FIG. 1 illustrates a high level overview of a system 100 for associating identifiers with virtual processes according to one embodiment of the present invention. A computer memory 101 includes user address space 103 and operating system address space 105. Multiple initialization processes 107 execute in user address space 103. Although FIG. 1 illustrates only two initialization processes 107 executing in user address space 103, it is to be understood that within a given computer memory 101, more than two initialization processes 107 can execute simultaneously.

Also executing in user address space are descendent processes 108, originating from the initialization processes 107. A descendent process 108 is a child process of an initialization process 107, pr a child process thereof, extended to any number of generations of subsequent child processes. Although FIG. 1 illustrates only two descendent processes 108 for each initialization process 107, it is to be understood that within a given computer memory 101, fewer or more than two descendent processes 108 per initialization process 107 can execute simultaneously.

Preferably, a data structure for storing associations 129 between executing processes (initialization processes 107 or descendent processes 108) and virtual processes

is inserted into the operating system 117. In one embodiment, the data structure is a mapping table 127, but in other embodiments other data structures are utilized, for example a linked list. In one embodiment, the mapping table 127 (or other data structure) is dynamically loaded into the operating system kernel 109, while the kernel 109 is active. In another embodiment, the mapping table 127 is stored in user address space 103. The maintenance and use of the mapping table 127 is discussed in detail below.

It is to be understood that a virtual process is not an actual process that executes in computer memory 101. Instead, the term "virtual process" describes a collection of associated functionality. For example, a virtual private server application is not actually a discrete process, but instead comprises a plurality of actual processes that together provide the desired functionality, thereby simulating the existence of a single server application executing on a dedicated physical host. Each actual process that performs some of the functionality of the virtual private server is a part of the virtual process. In FIG. 1 for example, initialization process 1 and descendent processes 1 and 2 (the processes descended from initialization process 1) comprise one virtual process, whereas initialization process 2 and descendent processes 3 and 4 comprise another.

In order to associate a specific identifier with each actual process that is a part of a virtual process, a separate system initialization process 107 is started for each virtual process. Normally, each process executing on a multitasking operating system is descended from a single system initialization process 107 that is started when the operating system 117 is booted. However, the present invention uses techniques described in detail below to start a separate system initialization process 107 for each virtual process . When each system initialization process 107 is started, an association

129 between the system initialization process 107 and the virtual process is stored in the mapping table 127. All additional processes that are descended from a given initialization process are thus identified with the virtual process associated with that initialization process.

In one embodiment, rather than starting a separate system initialization process 107 for each virtual process, a custom initialization process is started. In this embodiment, all processes that are a part of a specific virtual process are descended from the associated custom initialization process, and are associated with the virtual process with which the custom initialization process is associated. The exact functionality included in the custom initialization process is a design choice that can be made by a system administrator.

System calls 115 that generate child processes (for example, the UNIX® fork and clone functions) are intercepted so that the child processes can be associated with the virtual process with which the parent process is associated. A system call wrapper 111 is utilized in order to intercept system calls 115. In one embodiment, the system call wrapper 111 is dynamically loaded into the operating system kernel 109, while the kernel 109 is active. In another embodiment, the system call wrapper is loaded in user address space 103. The system call wrapper 111 is preferably in the form of object code, the functional features of which are described in detail below.

Pointers 114 to system calls 115 are located in an operating system call vector table 113. It is to be understood that the term "system call vector table" as used herein denotes an area in operating system address space 105 in which there are stored the addresses of system calls. In the UNIX® operating system, this part of the operating

system is called the "system call vector table," and that term is used in this specification. Other operating systems employ different terminology to denote the same system component. A system call vector table by any other name is still within the scope of the present invention.

A copy 116 is made of a pointer 114 to each system call 115 to be intercepted. These copies 116 of pointers 114 are preferably stored in operating system address space 105, but in an alternative embodiment are stored in user address space 103. Once the copies 116 have been made and saved, the pointers 114 in the system call vector table 113 to the system calls 115 to be intercepted are replaced with pointers 118 to the system call wrapper 111, such that when a system call 115 to be intercepted is made, the system call wrapper 111 executes instead. In one embodiment, this copying, storing, and replacing of pointers is performed by the system call wrapper 111. In other embodiments, copying, storing, and replacing of pointers is performed by a pointer management module executing in either operating system address space 105 or user address space 103 as desired. The pointer management module can either be a stand alone program, or a component of a larger application program as desired.

Executing alternative code when a system call 115 is made comprises intercepting the system call 115. The steps of inserting a system call wrapper 111 into the operating system 117, making a copy 116 of an operating system pointer 114 to a system call 115, and replacing the operating system pointer 114 with a pointer 118 to the system call wrapper 111 facilitate interception of a system call 115. When a system call 115 to be intercepted is made, the operating system 117 uses the pointer 118 in the system call vector table 113 to the system call wrapper 111 to execute the system call wrapper 111.

It is to be understood that only system calls 115 that create child processes need be intercepted, and thus only pointers 114 to system calls 115 to be intercepted are replaced with pointers 118 to the system call wrapper 111. Pointers 114 to system calls 115 which are not to be intercepted are not replaced. Thus, when a non-intercepted system call 115 is made, the system call 115 executes, not the system call wrapper 111.

The various initialization processes 107 and descendent processes 108 execute in user address space 103 under control of the operating system 117, and make system calls 115. When a process makes a system call 115 that creates a child process, the system call wrapper 111 reads the mapping table 127, and determines whether the process that made the system call (the parent of the child process being created) is associated with a virtual process. If so, the system call wrapper 111 uses the saved copy of the pointer 116 to execute the system call 115, allowing the creation of the child process. The system call wrapper 111 updates the mapping table 127, storing an association 129 between the newly created child process and the virtual process with which the process that made the system call is associated. Thus, all descendent processes 108 are associated with the virtual process with which their parent process is associated.

II. STARTING INITIALZIATION PROCESSES BY A MANAGER PROGRAM

FIG. 2 illustrates one embodiment of a system 200 for associating identifiers with virtual processes. In the embodiment illustrated by FIG. 2, the initialization processes 107 are started by a virtual process manager program 201 executing in user address space 103.

The virtual process manager program 201 modifies the operating system 117 of the computer to include the mapping table 127. Preferably, the manager program 201 loads the mapping table 127 into the kernel 109 of the operating system 117 while the kernel is active.

For each virtual process, the manager program 201 starts an initialization process 107 from which all other processes that are part of the virtual process will originate as descendent processes 108. Each time the manager program 201 starts an initialization process 107 for a virtual process, the manager program 201 stores, in the mapping table 127, an association 129 between the initialization process 107 and the appropriate virtual process. Subsequently, all additional processes that are part of the virtual process will be originated from the initialization process, and thus associated with the appropriate virtual process.

For example, suppose that the manager program 201 starts a first virtual private server. To do so, the manager process 107 starts an initialization process 107 for the virtual private server, storing an association 129 between the initialization process 107, and a virtual process identifier for the virtual private server. Additional processes that are part of the virtual private server originate from the initialization process 107, and are associated with the virtual process identifier of the virtual private server. The manager process 201 can proceed to start a second virtual private server by starting a separate initialization process 107, and associating the second initialization process 107 with a separate virtual process identifier for the second virtual private server. Consequently, all of the processes associated with the second virtual private server will be associated with

the appropriate virtual process identifier. In this manner, multiple virtual processes on the same physical computer are each associated with unique identifiers.

III. STARTING INITIALZIATION PROCESSES BY A LOADER PROGRAM

FIG. 3 illustrates another embodiment of a system 300 for associating identifiers with virtual processes. In the embodiment illustrated by FIG. 3, initialization processes 107 are not started by a manager program 201, but instead are loaded by a modified loader program 301.

A loader program is an operating system utility that is used to execute computer programs that are stored on static media. Typically, a loader program loads an executable image from static media into user address space 103 of computer memory 101, and then initiates execution of the loaded image by transferring execution to the first instruction thereof.

Like a standard loader program, the modified loader 301 loads executable images (in this case, initialization processes 107) from static media into user address space 103. Additionally, the modified loader program 301 stores, in the mapping table 127, an association 129 between the initialization process 107 being loaded and the appropriate virtual process. Thus, for each virtual process, an initialization process 107 is loaded by the modified loader program, and an association between the initialization process 107 and the virtual process is stored in the mapping table 127. Subsequently, additional processes that are part of the virtual process originate from the associated initialization process 107, and are thus associated with the virtual process as described above.

In another embodiment, the modified loader program 301 loads all processes that are part of each virtual process. In that embodiment, whenever the modified loader program 301 loads a process, the modified loader program 301 also stores, in the mapping table, an association 129 between the loaded process and the appropriate virtual process.

As will be understood by those familiar with the art, the invention may be embodied in other specific forms without departing from the spirit or essential characteristics thereof. Likewise, the particular naming of the modules, features, attributes or any other aspect is not mandatory or significant, and the mechanisms that implement the invention or its features may have different names or formats. Accordingly, the disclosure of the present invention is intended to be illustrative, but not limiting, of the scope of the invention, which is set forth in the following claims.